IDL_Guide Documentation

Release 0.9

us

Dec 10, 2019

CONTENTS

This package can be used to fit an Implicit Deep Learning (IDL) model for regression and classification purpose.

CHAPTER

ONE

INTRODUCTION

This package can be used to fit an Implicit Deep Learning (IDL) model for regression and classification purpose.

The IDL.fit function estimates a vector of parameters by applying successively gradient descents (see more at *Gradient Descents*) and dual ascent (see more at *Dual Ascents*).

1.1 Implicit Deep Learning

Given an input $u \in \mathbb{R}^n$, where n denotes the number of features, we define the implicit deep learning prediction rule $\hat{y}(u) \in \mathbb{R}^n$ with ReLU activation

$$\hat{y}(u) = Ax + Bu + c \qquad (1.1)$$

$$x = (Dx + Eu + f)_+ (1.1)$$

where $(.)_+ := \max(0, .)$ is ReLU activation, $x \in \mathbb{R}^h$ is called the hidden variable (h is the number of hidden features), $\Theta := (A, B, c, D, E, f)$ are matrices and vectors of appropriate size, they define the parameters of the model. The hidden variable x is implicit in the sense that there is in general no analytical formula for it, this is different from classic deep learning for which, given the model parameters, the hidden variables can be computed explicitly via propagation through the network.

1.2 Notation and definitions

We denote $\|.\|$ the eucledian norm, $\|.\|_2$ the corresponding norm (i.e. the spectral norm) and $\|.\|_F$ the Frobenius norm. $\mathbb{R}^n_+$ denotes the positive orthant of the vector space $\mathbb{R}^n$, $\mathbb{S}^n$ the set of real symmetric matrices of size n and $\mathbb{S}^n_+$ the cone of positive semi-definite matrices of size n. The transpose of a matrix or vector is denoted $.^T$ and elementwise product is denoted $\odot$. Given a differentiable function f from $\mathbb{R}^{n \times p}$ to $\mathbb{R}$ we define the scalar by matrix partial derivative in denominator layout convention as

$$\frac{\partial f}{\partial A} = \nabla_A f = \begin{bmatrix} \frac{\partial f}{\partial A_{1,1}} & \cdots & \frac{\partial f}{\partial A_{1,p}} \\ \vdots & \ddots & \vdots \\ \frac{\partial f}{\partial A_{n,1}} & \cdots & \frac{\partial f}{\partial A_{n,p}} \end{bmatrix} \in \mathbb{R}^{n \times p}.$$

We say that a function $(x, y) \to f(x, y)$ with seperable domain of definition $\mathcal{X} \times \mathcal{Y}$ is bi-convex in (x, y), if for all $x \in \mathcal{X}$, the function $y \to f(x, y)$ is convex and for all $y \in \mathcal{Y}$ the function $x \to f(x, y)$ is convex. We say that a function is smooth if it is differentiable and its gradient is Lipschitz continious. We say that f is bi-smooth if it is smooth in x given y and vice-versa. An example of bi-smooth and bi-convex function is $(x, A) \to x^T Ax, A \in \mathbb{S}^n_+$.

1.3 Well-posedness

We say that matrix D is well-posed for (1.1) if there exists a unique solution $x = (Dx + \delta)_+ \forall \delta \in \mathbb{R}^h$. Using the fact that ReLU is 1-Lipschitz we have for $x_1, x_2 \in \mathbb{R}^h$

$$\|(Dx_1 + \delta)_+ - (Dx_2 + \delta)_+\| \leq \|D(x_1 - x_2)\| \leq \|D\|_2 \|x_1 - x_2\|.$$

If $\|D\|_2 < 1$ we have that the map $x \rightarrow (Dx + \delta)_+$ is a strict contraction. In that case, Banach's contraction mapping theorem applies, showing that the equation $x = (Dx + \delta)_+$ has a unique solution. In that case, a solution x can be computed via the Picard iterations

$$x^{k+1} = (Dx + \delta), k = 1, 2, \cdots .$$

Note that $\|D\|_2 < 1$ is only a sufficient condition for well-posedness. Nevertheless this is the only condition we will consider in this article.

1.4 Loss Functions

** section 3.2 equation 4 ** + Classification loss (see more at *Learning Process*)

1.5 Description of the learning process

(see more at *Formulation for IDL*)

1.6 Description of the prediction process

(see more at *Predicting*)

1.7 Setup

TODO

The package is compatible with Python version 3 or higher only. The user is expected to have installed cvxpy before running the package. Go to … for more information.

1. Switch to a proper directory and then type:

```
git clone + https://github.com/...
```

FORMULATION FOR IDL

See *Citing* for in-depth explanation

2.1 Problem formulation

Let us consider the input and output data matrices $U = [u_1, \cdots, u_m] \in \mathbb{R}^{n \times m}, Y = [y_1, \cdots, y_m] \in \mathbb{R}^{p \times m}$ with m being the number of datapoints - the corresponding optimization regression problem (i.e. with squared error loss) reads

$$\begin{aligned} \min_{X,\Theta} \quad & \mathcal{L}(Y, [\Theta, X]) := \frac{1}{2m}\|AX + BU + c1_m^T - Y\|_F^2 \\ \text{s.t.} \quad & X = (DX + EU + f1_m^T)_+ \; (2.1) \\ & \|D\|_2 < 1, \; (2.1) \end{aligned} \tag{2.1}$$

where 1_m is a column vector of size m consisting of ones. For clarity we have highlighted in blue the optimization variables. The non-convexity of this problem arises from the nonlinear implicit constraint and the matrix product terms AX and DX. In practice we replace the constraint $\|D\|_2 < 1$ by the closed convex form $\|D\|_2 \leq 1 - \epsilon$, where $\epsilon > 0$ is small.

2.2 Fenchel Divergence Lagrangian Relaxation

Using Fenchel-Young inequality, it can be shown that the equation $x = (Dx + Eu + f)_+$ is equivalent to

$$\begin{cases} \mathcal{F}(x, Ax + Bu + c) = 0 \\ x \geq 0 \end{cases} \tag{2.1}$$

with the Fenchel Divergence $\mathcal{F}$ defined by

$$\mathcal{F}(x_1, x_2) := \frac{1}{2}x_1 \odot x_1 + \frac{1}{2}(x_2)_+ \odot (x_2)_+ - x_1 \odot x_2.$$

We use the term divergence because by construction $\mathcal{F}(x_1, x_2) \geq 0 \forall x_1, x_2 \in \mathbb{R}_+^h \times \mathbb{R}^h$. Given $X = [x_1, \cdots, x_m]$ and $Z = [z_1, \cdots, z_m]$ we write

$$\mathcal{F}(X, Z) := \frac{1}{m}\sum_{i=1}^{m}\mathcal{F}(x_i, z_i).$$

A Lagrangian relaxation approach to solving (2.1) problem using the implicit constraint formulation (2.1) consists in solving given a dual variable $\lambda \in \mathcal{R}_+^h$

$$\begin{aligned} \min_{X \geq 0,\Theta} \quad & \mathcal{L}(Y, [\Theta, X]) + \lambda^T\mathcal{F}(X, DX + EU + f1_m^T) \\ \text{s.t.} \quad & \|D\|_2 < 1, \; (2.2) \end{aligned} \tag{2.2}$$

This problem is bi-smooth in $[\Theta, X]$, but it is not convex or bi-convex. Nevertheless we can make it bi-convex with extra conditions on Θ as shown in the next section.

2.3 Linear matrix inequality parameter constraints for bi-convexity

Let us define $\Lambda = diag(\lambda) \in \mathbb{S}^h_+$

Theorem 1. *Problem* (2.2) *is bi-convex in* $[\Theta, X]$ *if we impose one of the two following feasible linear matrix inequalities (LMI)*

$$\Lambda - (\Lambda D + D^T \Lambda) \in \mathbb{S}^h_+ \tag{2.2}$$

$$\Lambda + A^T A - (\Lambda D + D^T \Lambda) \in \mathbb{S}^h_+ \tag{2.3}$$

Proof. The loss term $\mathcal{L}(Y, [\Theta, X])$ is already bi-convex in (Θ, X), but it is not the case for the Fenchel Divergence term $\lambda^T \mathcal{F}(X, DX + EU + f1_m^T)$, which is not convex in X in the general case. A sufficient condition for this term to be convex in X given Θ is for the following function

$$x \rightarrow \lambda^T(\frac{1}{2} x \odot x - x \odot Dx) = \frac{1}{2} x^T (\Lambda - (\Lambda D + D^T \Lambda))x,$$

to be convex. This term is convex in x if the LMI (2.2) is satisfied. Now the second LMI similarly arises by leveraging the fact that we can also use the term in the loss to make the objective convex in x. Indeed the objective function of (2.1) is convex in x if

$$x \rightarrow \frac{1}{2} x^T A^T A x + \frac{1}{2} x^T (\Lambda - (\Lambda D + D^T \Lambda))x,$$

is convex, which corresponds to LMI (2.3). It might not be obvious that (2.3) is actually an LMI, but using Schur complement we can prove it is equivalent to

$$-\begin{bmatrix} I_p & A \\ A^T & \Lambda D + D^T \Lambda - \Lambda \end{bmatrix} \in \mathbb{S}^{p+h}_+.$$

If D satisfies (2.2) then it satisfies (2.3). We imediately have that $D = \delta I_n$ with $\delta \leq \frac{1}{2}$ satisfies (2.2) (and $\|D\|_2 \leq 1 - \epsilon$). Which proves that both LMIs are feasible.

2.4 Bi-convex Formulation

From this proof, the problem formulation reads

$$\begin{aligned} \min_{X \geq 0, \Theta} \quad & \mathcal{L}(Y, [\Theta, X]) + \lambda^T \mathcal{F}(X, DX + EU + f1_m^T) & (2.4) \\ \text{s.t.} \quad & \|D\|_2 \leq 1 - \epsilon \, (2.4) \\ & \Lambda + A^T A - (\Lambda D + D^T \Lambda) \in \mathbb{S}^h_+, \, (2.4) \end{aligned}$$

this problem is well-posed -feasible solutions exist- and bi-smooth.

CHAPTER THREE

LEARNING PROCESS

** section 3.4 algo in the end **

class `IDL.`**`IDLModel`**(*hidden_variables=1*, *dual_learning_rate=0.1*, *tol_fenchtel=0.1*, *inner_tol=0.001*, *starting_lambda=None*, *initialization_theta=None*, *random_state=0*, *early_stopping=True*, *verbosity=True*)

Implementation of the Scikit-Learn API for Implicit Deep Learning

Parameters

- **`hidden_variables`** – int number of hidden variables of the vector X (see
- **`dual_learning_rate`** – float Positive float, Dual learning rate (IDL's "alpha")
- **`tol_fenchel`** – float Positive float, Fenchel tolerance threshold for dual's update (IDL's "alpha")
- **`inner_tol`** – float Positive float, tolerance threshold for early stopping in the gradient descents with respect to theta
- **`verbosity`** – bool Verbosity of the training process.
- **`random_state`** – int Random number seed for initialization.

Full documentation of parameters can be found here: https://github.com/GuillaumeGoujard/IDLEstimator/blob/master/docs/source/sections/introduction.rst.

`fit`(*X*, *y*, *rounds_number=100*, *verbose=True*, *type_of_training='two_loops'*)

Fit IDL Model

Parameters

- **`X`** – array_like Feature matrix
- **`y`** – array_like Labels
- **`rounds_number`** – int Maximum rounds number in the outer loop
- **`verbose`** – bool
- **`type_of_training`** – string Two types of training :
  - "two_loops" : RECOMMENDED, we optimize theta and X variables and then we do one step of dual ascent.
  - "one_loop" : one iteration is going to successively operate one step of gradient descent and one step

  of dual ascent

Returns self : object Returns self.

CHAPTER FOUR

GRADIENT DESCENTS

4.1 Block coordinate descent and first order methods

As problem (2.4) is bi-convex, a natural strategy is the use of block coordinate descent (BCD): alternating optimization between Θ. BCD corresponds to the following algorithm,

for $k = 1, 2, \cdots$ **do**

$$\Theta^k \in \text{argmin}_\Theta \frac{1}{2m}\|AX^{k-1} + BU + c1_m^T - Y\|_F^2 + \lambda^T \mathcal{F}(X^{k-1}, DX^{k-1} + EU + f1_m^T) \tag{4.1}$$

$$\Lambda + A^T A - (\Lambda D + D^T \Lambda) \in \mathbb{S}_+^h \tag{4.2}$$

$$\|D\|_2 \leq 1 - \epsilon \tag{4.3}$$

$$X^k \in \text{argmin}_{X \geq 0} \frac{1}{2m}\|A^k X + B^k U + c^k 1_m^T - Y\|_F^2 + \lambda^T \mathcal{F}(X, D^k X + E^k U + f^k 1_m^T) \tag{4.4}$$

end

In practice such updates might be to heavy computationally as the number of datapoints m increase, or as the model size increases (i.e. h, n or p). Instead we propose to do block coordinate projected gradient updates. This method is also considered to be better at avoiding local minima. Let us denote

$$\mathcal{G}(\Theta, X) := \mathcal{L}(Y, [\Theta, X]) + \lambda^T \mathcal{F}(X, DX + EU + f1_m^T)$$

In the remainder of this section we derive the gradients $\nabla_\Theta \mathcal{G}(\Theta, X), \nabla_X \mathcal{G}(\Theta, X)$ and corresponding 'optimal' step-sizes using the Lipschitz coefficients of the gradients- which is the advantage of having a bi-smooth optimization problem. Note that the objective $\mathcal{G}$, given X is separable in $\Theta_1 := (A, B, c)$ and $\Theta_2 := (D, E, f)$. Using scalar by matrix calculus

$$\begin{cases} \nabla_A \mathcal{G}(\Theta, X) = \Omega(A, B, c) X^T \in \mathbb{R}^{p \times h} \\ \nabla_B \mathcal{G}(\Theta, X) = \Omega(A, B, c) U^T \in \mathbb{R}^{p \times n} \\ \nabla_c \mathcal{G}(\Theta, X) = \Omega(A, B, c) 1_m \in \mathbb{R}^p \end{cases},$$

with $\Omega(A, B, c) := \frac{1}{m}(AX + BU + c1_m^T - Y) \in \mathbb{R}^{p \times m}$. Hence we can show that a Lipschitz constant for the gradient is given by

$$L_{\Theta_1}(X) := \frac{1}{m} \max(m, \|X\|_2^2, \|U\|_2^2, \|XU^T\|_2),$$

and the 'optimal' step-size for gradient descent is then simply given by

$$\alpha_{\Theta_1}(X) := \frac{1}{L_\Theta(X)}.$$

Regarding the gradient with respect to Θ_2, we have

$$\begin{cases} \nabla_D \mathcal{G}(\Theta, X) = \Omega(D, E, f, \Lambda) X^T \in \mathbb{R}^{h \times h} \\ \nabla_E \mathcal{G}(\Theta, X) = \Omega(D, E, f, \Lambda) U^T \in \mathbb{R}^{h \times n} \\ \nabla_f \mathcal{G}(\Theta, X) = \Omega(D, E, f, \Lambda) 1_m \in \mathbb{R}^{h} \end{cases},$$

with $\Omega(D, E, f, \Lambda) := \frac{\Lambda}{m}\left((DX + EU + f1_m^T)_+ - X\right) \in \mathbb{R}^{h \times m}$, we can show that a Lipschitz constant for the gradient is

$$L_{\Theta_2}(X) := \frac{\lambda_{\text{max}}}{m} \max(m, \|X\|_2^2, \|U\|_2^2, \|X\|_2 \|U\|_2),$$

where $\lambda_{\text{max}} = \max_{j \in \{1, \cdots, h\}} \lambda_j$. We can then similarly define an 'optimal' step-size $\alpha \Theta_2$. We have that

$$\nabla_X \mathcal{G}(\Theta, X) = \frac{1}{m}\left\{A^T(AX + BU + c1_m^T) + (\Lambda - \Lambda D - D^T \Lambda)X + D^T \Lambda (DX + EU + f1_m^T)_+ - \Lambda(EU + f1_m^T)\right\}.$$

A Lipschitz constant for this gradient is

$$L_X(\Theta) = \frac{1}{m}(\|A^T A + \Lambda - \Lambda D + D^T \Lambda\|_2 + \lambda_{\text{max}} \|D\|_2^2).$$

We can then take the step-size $\alpha_X(\Theta) = \frac{1}{L_X(\Theta)}$. We propose the following block coordinate projected gradient scheme (BC-gradient) to find a candidate solution to :eq:'eq_7. We denote compactly the convex set

$$\mathcal{S}_\Theta := \{\Theta | \Lambda + A^T A - (\Lambda D + D^T \Lambda) \in \sim_+^h, \|D\|_2 \leq 1 - \epsilon\}$$

and $\mathcal{P}_{\mathcal{S}_\Theta}$ the corresponding convex projection

for $k = 1, 2, \cdots$ **do**

$$\Theta^k = \mathcal{P}_{\mathcal{S}_\Theta}\left(\Theta^k - \alpha_\Theta(X^{k-1}) \nabla_\Theta \mathcal{G}(\Theta^{k-1}, X^{k-1})\right) \tag{4.5}$$

$$X^k = \left(X^{k-1} - \alpha_X(\Theta^k) \nabla_X \mathcal{G}(\Theta^k, X^{k-1})\right) \tag{4.6}$$

end

4.2 Bi-Convexity of the Loss function

TODO

4.3 Gradient Descents

TODO

4.4 Code for calculating Gradient Descent

`utilities.GradientDescents.`**`gradient_descent_theta`**`(`*theta*, *X*, *U*, *Y*`)`

Returns the gradient of theta

Parameters

- **`theta`** – a dictionary
- **`X`** – hidden variables
- **`U`** – input data
- **`Y`** – output data

Returns grad_theta: dictionary containing gradients of elemnts in theta

CHAPTER

FIVE

DUAL ASCENTS

5.1 Dual methods

We propose the following schemes to find an appropriate dual variable λ. Let $\epsilon > 0$ be a precision parameter for the implicit constraint, i.e. such that we would have

$$\mathcal{F}(X, DX + EU + f1_m^T) \leq \epsilon$$

We start with $\lambda = 0$ and we solve the two following separate problems

$$\min_{X>0,A,B,c} \frac{1}{m} \|AX + BU + c1_m^T - Y\|_F^2$$

and then

$$\min_{D,E,f} 1_h^T \mathcal{F}(X, DX + EU + f1_m^T).$$

If $\mathcal{F}^* := \mathcal{F}(X, DX + EU + f1_m^T) < \epsilon I_h$ then we stop there. Otherwise, we do one of the two following 'dual updates'

5.2 Dual ascent conditional on Fenchel Divergence

$$\lambda \leftarrow \lambda + \alpha \mathcal{F}^* \odot 1\{\mathcal{F}^* \geq \epsilon I_h\}, \tag{5.1}$$

where $\alpha > 0$ is a step-size. Note that here we only update the components of λ for which the corresponding Fenchel divergence is more than ϵ. We then proceed to solve (2.4) using previously discussed methods and iterate. Alternatively, if the BC-gradient method is used, we can do a dual update after each BC-gradient update.

5.3 Dual variable update conditional on loss

We start with $\lambda = \epsilon I_h$. Given (Θ, X), we define the unique $\bar{X}$ such that the implicit constraint is enforced given Θ

$$\bar{X} = (DX + EU + f1_m^T)_+.$$

We then define $\Delta X := X - \bar{X}$. We can compute in close form the error on the loss due to the implicit constraint violation

$$\Delta\mathcal{L} := \mathcal{L}(Y, [\Theta, \bar{X}]) - \mathcal{L}(Y, [\Theta, X]) \tag{5.2}$$

$$= \frac{1}{2m}\left(\|A\Delta X\|_F^2 + Tr(\Omega, A\Delta X)\right) \tag{5.3}$$

with $\Omega := BU + c1_m^T$. We can write this error as a sum of contributions with respect to each hidden variable components $j \in \{1, \cdots, h\}$

$$\Delta\mathcal{L} = \sum_{j=1}^{h} \left\{ \Delta\mathcal{L}_j := \frac{1}{m} A_j^T \left(\frac{1}{2} A\Delta X + \Omega \right) \Delta X_j^T \right\},$$

where $A_j \in \mathbb{R}^h$ is the j^{th} column of A and $\Delta X_j \in \mathbb{R}^{1\times m}$ is the j^{th} row of ΔX. The objective of this dual update is to achieve an error on the loss that is smaller than a fraction $\eta \in (0, 1)$ of the loss

$$\frac{\Delta\mathcal{L}}{\mathcal{L}(Y, [\Theta, \bar{X}])} \leq \eta.$$

In order to update each component of the dual variable, we propose the following update. Given $j \in \{1, \cdots, h\}$ if

$$\frac{(\Delta\mathcal{L}_j)_+}{\mathcal{L}(Y, [\Theta, \bar{X}])} \geq \frac{\eta}{h},$$

then we do the update

$$\lambda_j \to \beta\lambda_j,$$

with $\beta > 1$ a hyperparameter.

5.4 Code for calculating Dual Ascents

CHAPTER SIX

PREDICTING

** section 2 Picard iterations **

class IDL.**IDLModel**(*hidden_variables=1*, *dual_learning_rate=0.1*, *tol_fenchtel=0.1*, *inner_tol=0.001*, *starting_lambda=None*, *initialization_theta=None*, *random_state=0*, *early_stopping=True*, *verbosity=True*)

Implementation of the Scikit-Learn API for Implicit Deep Learning

Parameters

- **hidden_variables** – int number of hidden variables of the vector X (see
- **dual_learning_rate** – float Positive float, Dual learning rate (IDL's "alpha")
- **tol_fenchel** – float Positive float, Fenchel tolerance threshold for dual's update (IDL's "alpha")
- **inner_tol** – float Positive float, tolerance threshold for early stopping in the gradient descents with respect to theta
- **verbosity** – bool Verbosity of the training process.
- **random_state** – int Random number seed for initialization.

Full documentation of parameters can be found here: https://github.com/GuillaumeGoujard/IDLEstimator/blob/master/docs/source/sections/introduction.rst.

predict(*X*, *k_iterations=10000*)

Predicting function. :param X: array-like

The input sample.

Parameters **k_iterations** – int Maximum number of Picard iterations

Returns y: array-like Returns a prediction array.

CHAPTER SEVEN

EXAMPLES

7.1 Classification : MNIST

7.2 Regression : Boston Housing

CHAPTER EIGHT

CITING

- "Implicit Deep Learning" Laurent El Ghaoui, Fangda Gu, Bertrand Travacca, Armin Askari, arXiv:1908.06315, 2019.
- "Bi-convex Implicit Deep Learning" Bertrand Travacca, October 2019

CHAPTER NINE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX